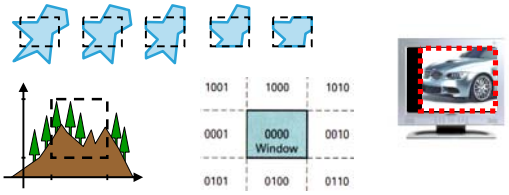
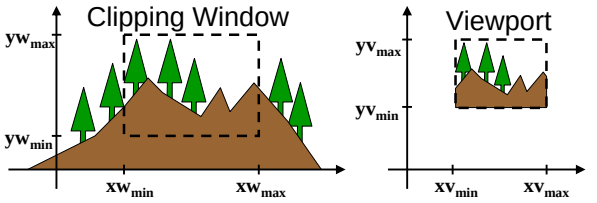
2D Viewing



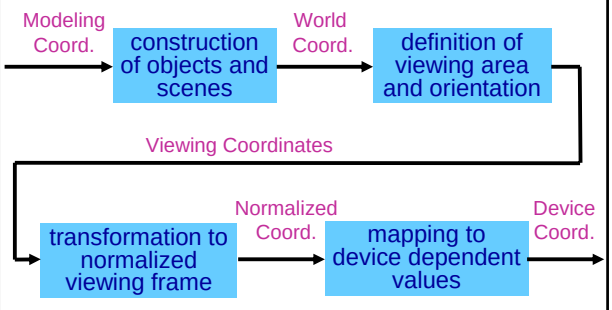
Viewing Pipeline: Window-Viewport Transf.

- clipping window: what to display
- viewport: where to be viewed
- translation, rotation, scaling, clipping,...



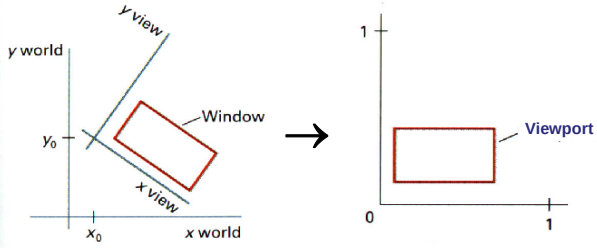
world coordinates *viewport coordinates*

2-dim. Viewing-Transformation Pipeline



WC → VC Transformation

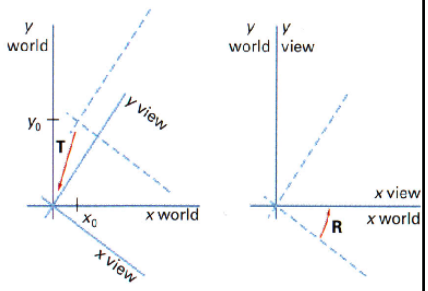
setting up a rotated world window in viewing coordinates and the corresponding normalized-coordinate viewport



World Coordinates **Viewing Coordinates**

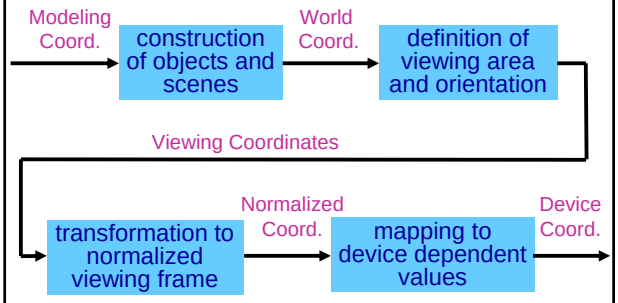
Viewing Coordinate Reference Frame

A viewing-coordinate frame is moved into coincidence with the world frame in two steps:



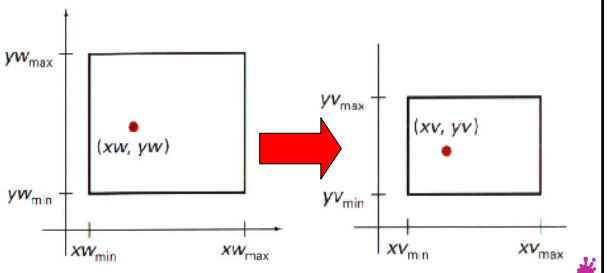
(a) translate the viewing origin to the world origin
(b) rotate to align the axes of the two systems.

2-dim. Viewing-Transformation Pipeline



Window - Viewport Transform (1)

point (xw,yw) in a designated window is mapped to viewport coordinates (xv,yv) so that relative positions in the two areas are the same.

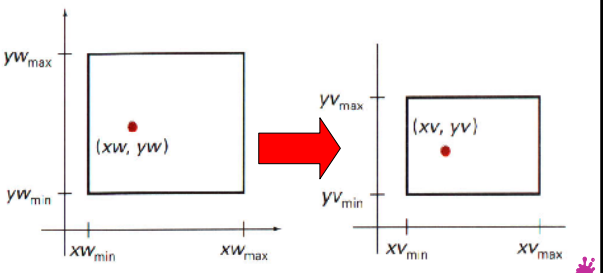


Window - Viewport Transform (2)

linear in x and y

$(xw_{min}/yw_{min}) \rightarrow (xv_{min}/yv_{min})$

$(xw_{max}/yw_{max}) \rightarrow (xv_{max}/yv_{max})$



Window - Viewport Transform (3)

$$xw = xw_{min} + \lambda \cdot (xw_{max} - xw_{min}) \quad \text{where } 0 \le \lambda \le 1$$
$$xv = xv_{min} + \lambda \cdot (xv_{max} - xv_{min})$$

$$\lambda = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$xv = xv_{min} + \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \cdot (xv_{max} - xv_{min})$$

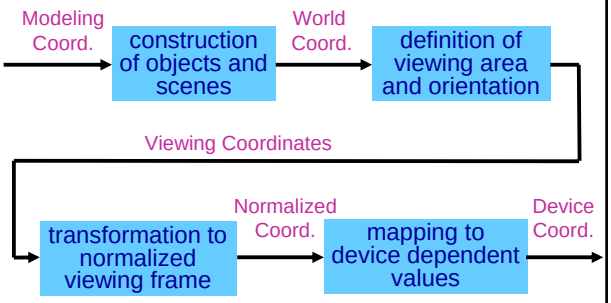
t_x $\qquad$ s_x

$$= xv_{min} - \frac{xw_{min} \cdot (xv_{max} - xv_{min})}{xw_{max} - xw_{min}} + xw \cdot \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

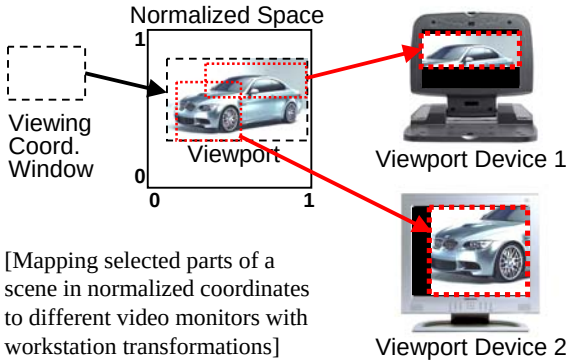
window-viewport transformation:

$$xv = s_x \cdot xw + t_x \qquad yv = s_y \cdot yw + t_y$$

2-dim. Viewing-Transformation Pipeline



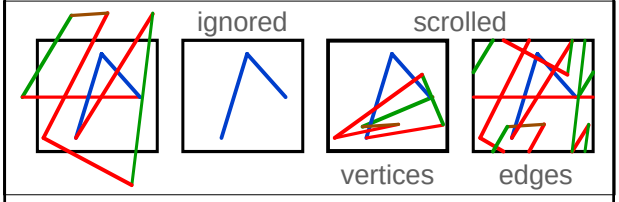
Workstation Transformation



[Mapping selected parts of a scene in normalized coordinates to different video monitors with workstation transformations]

Clipping

- partly visible or completely invisible parts
- must not be ignored and must not be drawn



- ⇒ must be cut off (if possible in world coordinates)

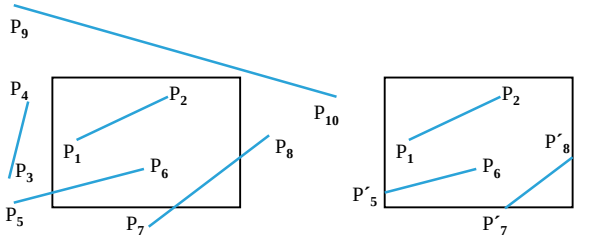
Clipping Operations

- remove objects outside a clip window
  - clip window: rectangle, polygon, curved boundaries
  - applied in world or viewing coordinates
  - combined with scan conversion
  - objects to clip: points, lines, polygons, curves, text, ...

3 Possibilities for Clipping

- *analytically* = in world coordinates
  - reduces WC → DC transformations
- *during raster conversation*
  = as part of the rasterization algorithm
  - efficient for complex primitives
- *pixel by pixel* test
  - biggest effort, very primitive algorithm

Line Clipping (1)



before clipping after clipping

[line clipping against a rectangular clip window]

Line Clipping (2)

- goals
  - eliminate simple cases fast
  - avoid intersection calculations

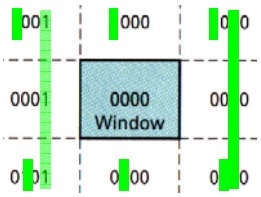
for endpoints (x_0,y_0), (x_{end},y_{end})
intersect parametric representation

$$x = x_0 + u\cdot(x_{end} - x_0)$$
$$y = y_0 + u\cdot(y_{end} - y_0)$$

with window borders:
intersection $\Leftrightarrow 0 < u < 1$

Cohen-Sutherland Line Clipping

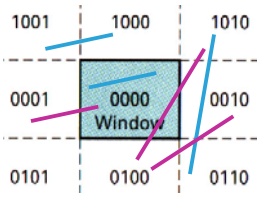
- assignment of region codes to line endpoints
  - bit1: left
  - bit2: right
  - bit3: below
  - bit4: above



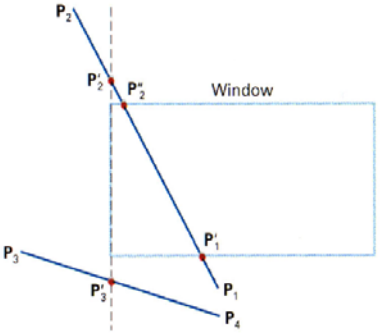
binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle

Cohen-Sutherland Line Clipping

- "or" of codes of both points = 0000 ⇒ line entirely ***visible***
- "and" of codes of both points ≠ 0000 ⇒ line entirely ***invisible***
- all others ⇒ ***intersect!***



Cohen-Sutherland Line Clipping



[lines extending from one coordinate region to another may pass through the clip window, or they may intersect clipping boundaries without entering the window]

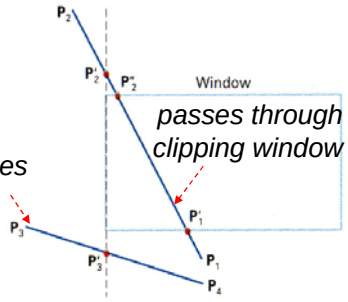
Cohen-Sutherland Line Clipping

- remaining lines
  - intersection test with bounding lines of clipping window
  - left, right, bottom, top
  - discard an outside part
  - repeat intersection test up to four times

vertical: $y = y_0 + m(xw_{min} - x_0)$, $y = y_0 + m(xw_{max} - x_0)$
horiz.: $x = x_0 + (yw_{min} - y_0)/m$, $x = x_0 + (yw_{max} - y_0)/m$

Cohen-Sutherland Line Clipping

intersects boundaries without entering clipping window

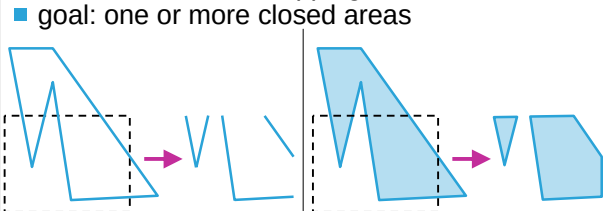


passes through clipping window

vertical: $y = y_0 + m(xw_{min} - x_0)$, $y = y_0 + m(xw_{max} - x_0)$
horiz.: $x = x_0 + (yw_{min} - y_0)/m$, $x = x_0 + (yw_{max} - y_0)/m$

Polygon Clipping

- modification of line clipping
- goal: one or more closed areas

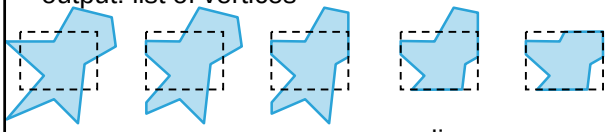


[display of a polygon processed by a line-clipping algorithm]

[display of a correctly clipped polygon]

Sutherland-Hodgman Polygon Clipping

- processing polygon boundary as a whole against each window edge
- output: list of vertices

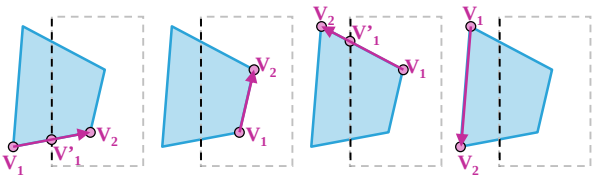


original polygon | clip left | clip right | clip bottom | clip top

clipping a polygon against successive window boundaries

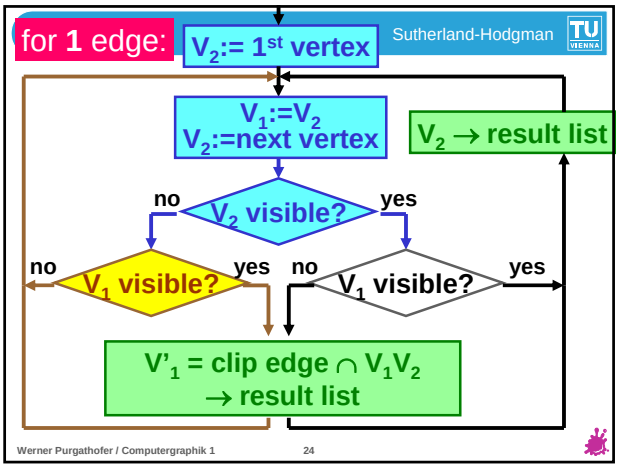
Sutherland-Hodgman Polygon Clipping

- four possible edge cases

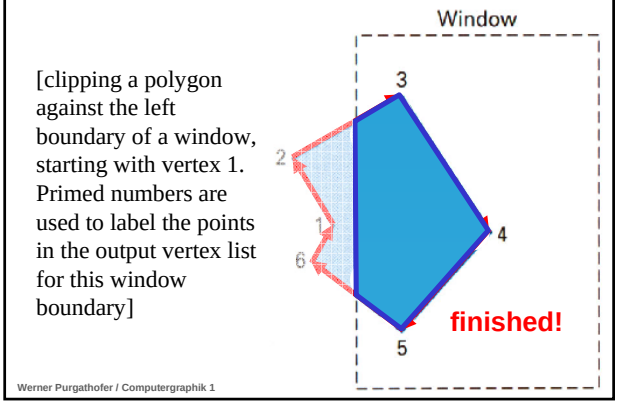


out → in	in → in	in → out	out → out
output: V'_1, V_2	V_2	V'_1	no output

successive processing of pairs of polygon vertices against the left window boundary



Sutherland-Hodgman Polygon Clipping



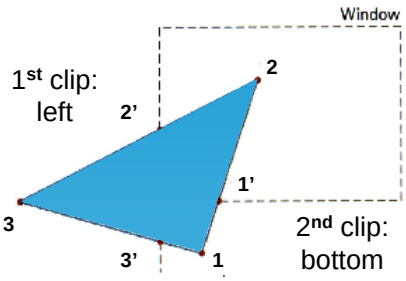
[clipping a polygon against the left boundary of a window, starting with vertex 1. Primed numbers are used to label the points in the output vertex list for this window boundary]

Polygon Clip: Combination of 4 Passes

- the polygon is clipped against each of the 4 borders separately,
  that would produce 3 intermediate results.
- by calling the 4 tests *recursively*,
  (or by using a clipping pipeline)
  every result point is immediately processed on, so that only *one* result list is produced

Sutherland-Hodgman Clipping Example

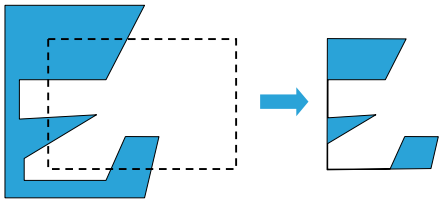
- pipeline of boundary clippers to avoid intermediate vertex lists



[Processing the polygon vertices through a boundary-clipping pipeline. After all vertices are processed through the pipeline, the vertex list for the clipped polygon is {1', 2, 2', 2"}]

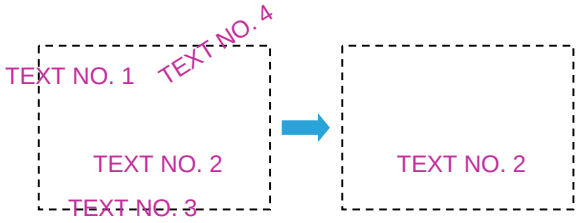
Sutherland-Hodgman Polygon Clipping

- extraneous lines for concave polygons:
  - split into separate parts or
  - final check of output vertex list



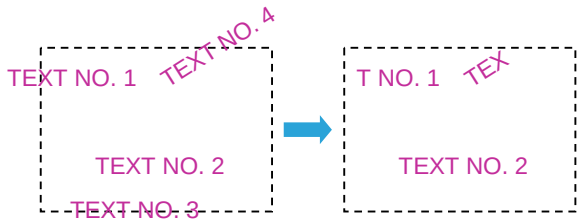
[clipping the concave polygon with the Sutherland-Hodgeman clipper produces two connected areas]

Text Clipping (1)



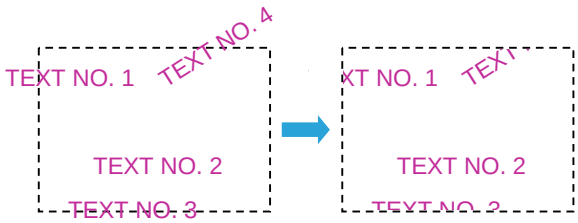
1. text clipping using a bounding rectangle about the entire string

Text Clipping (2)



2. text clipping using a bounding rectangle about individual characters

Text Clipping (3)



3. text clipping performed on the components of individual characters

Summary: Clipping

- Cohen-Sutherland line clipping
- Hodgman-Sutherland polygon clipping
- text clipping